Training and Optimization of Neural Networks and Generative Adversarial Networks over Distributed Resource

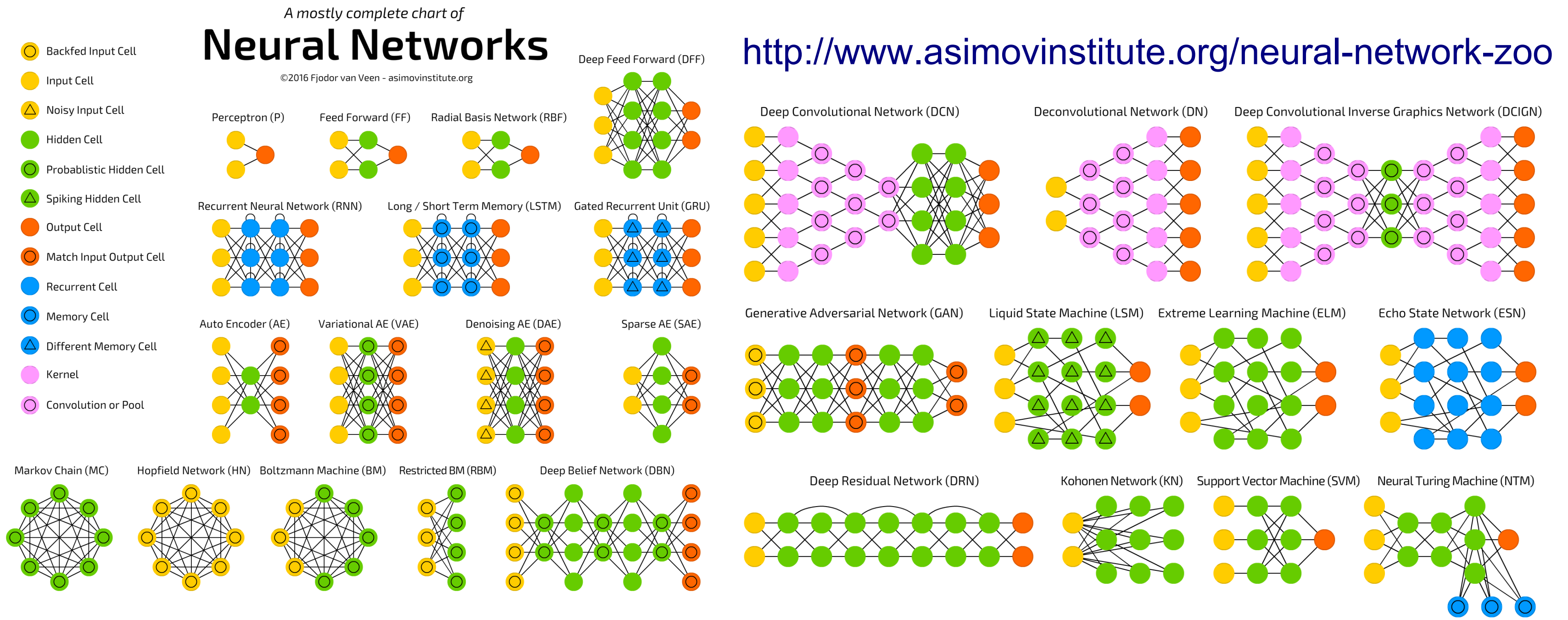
J-R. Vlimant, with many others V. Loncar, F. Pantaleo, T. Nguyen, M. Pierini, A. Zlokapa, S. Valecorsa, ...

Outline

- ANN and GAN
- Training workload parallelization
- Hyper-parameters optimization

Artificial Neural Network

Artificial Neural Network

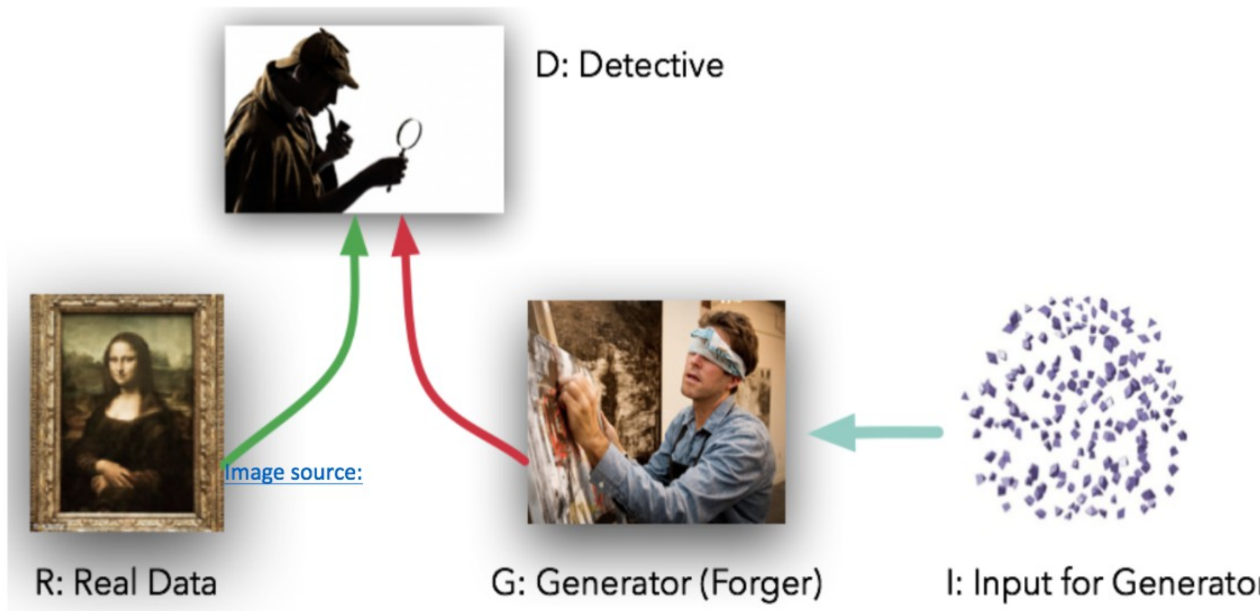


http://www.asimovinstitute.org/neural-network-zoo

- Large number of parameters
- Efficiently adjusted with stochastic gradient descent
- The more parameters, the more data required
- Training to convergence can take minutes to several days, ...

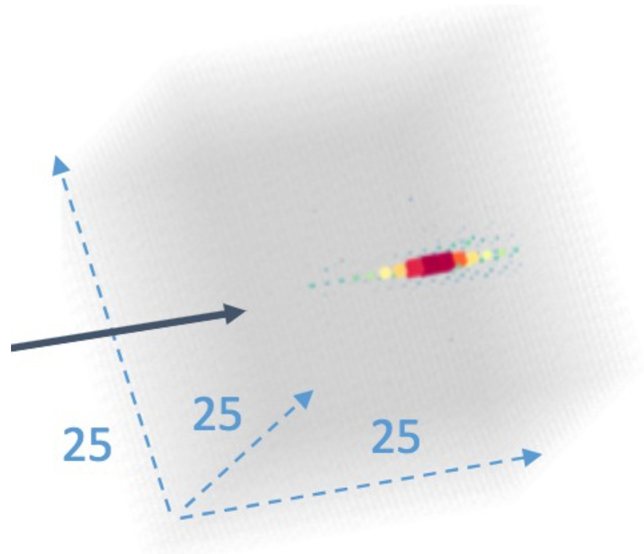
Generative Adversarial Network

A Forger's Game



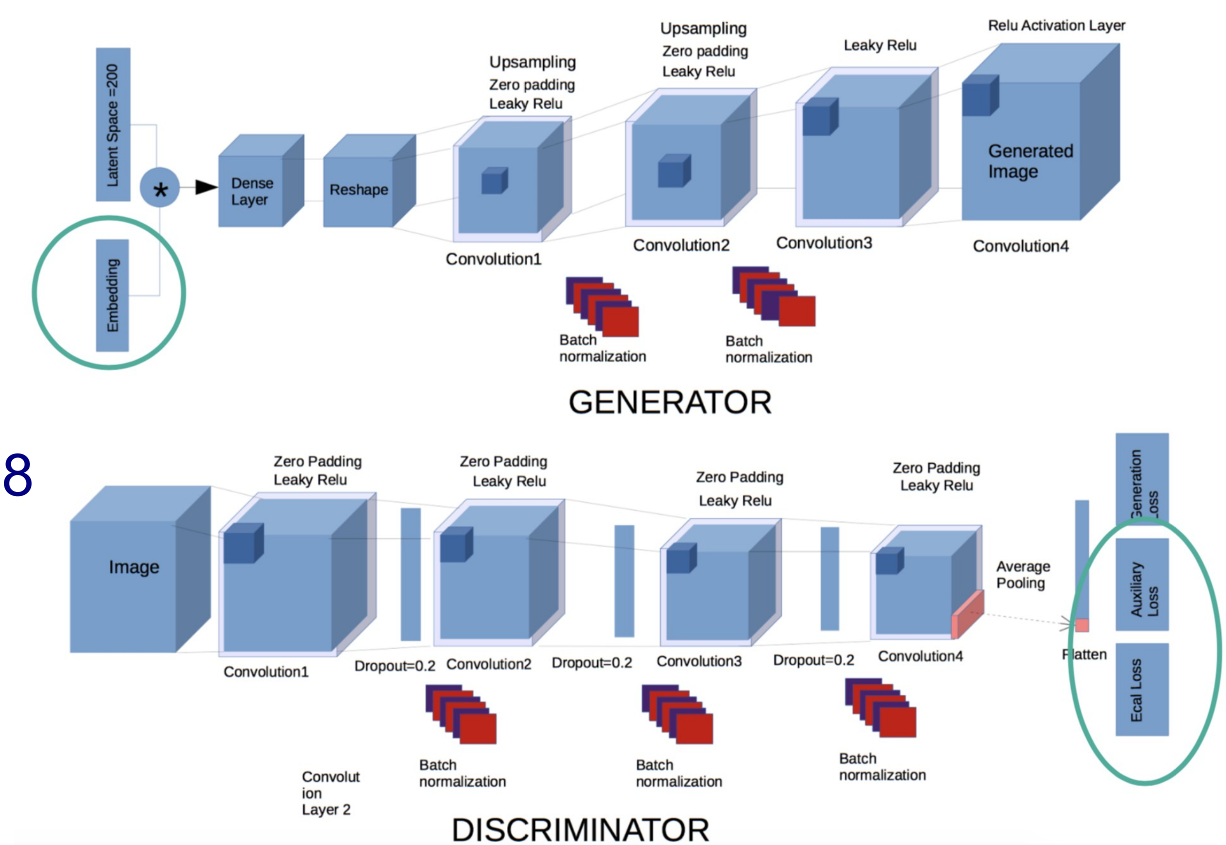
- Constructed from two artificial neural networks
- Two concurrent gradient descent procedure
- Training to convergence can take minutes to several days, ...

GAN for CLIC

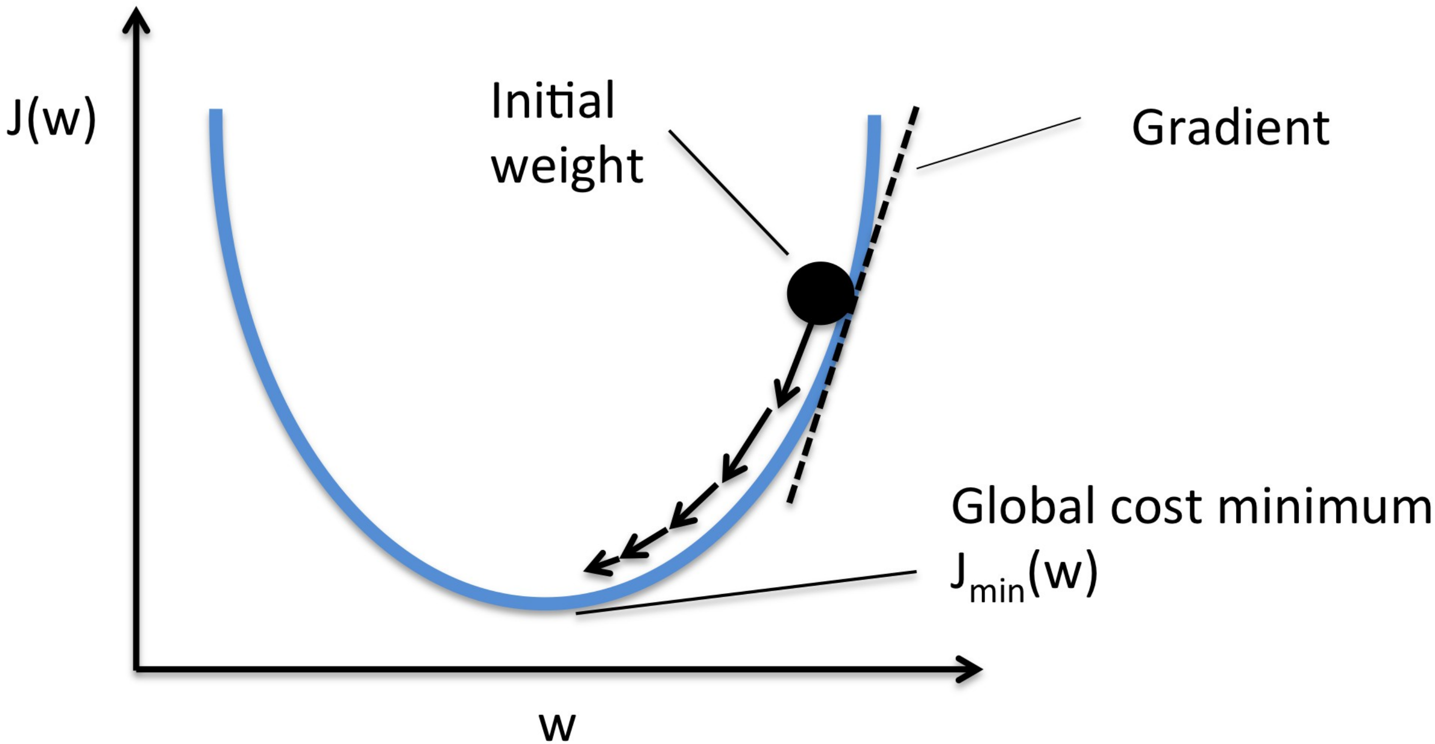


“Images” are 3D : energy deposition in a highly granular calorimeter
http://cds.cern.ch/record/2254048

Generative adversarial network architecture for CLIC 3D dataset



Training Artificial Neural Networks



- ANN and associated loss function have fully analytical formulation and are differentiable with respect to model parameters
- Gradient evaluated over batch of data
  - Too small : very noisy and scattering
  - Too large : information dilution and slow convergence

Distributed Training

Parallelism Overview

- ➔ **Data distribution**

  Compute the gradients on several batches independently and update the model synchronously or not. **Applicable to large dataset**

- ➔ **Gradient distribution**

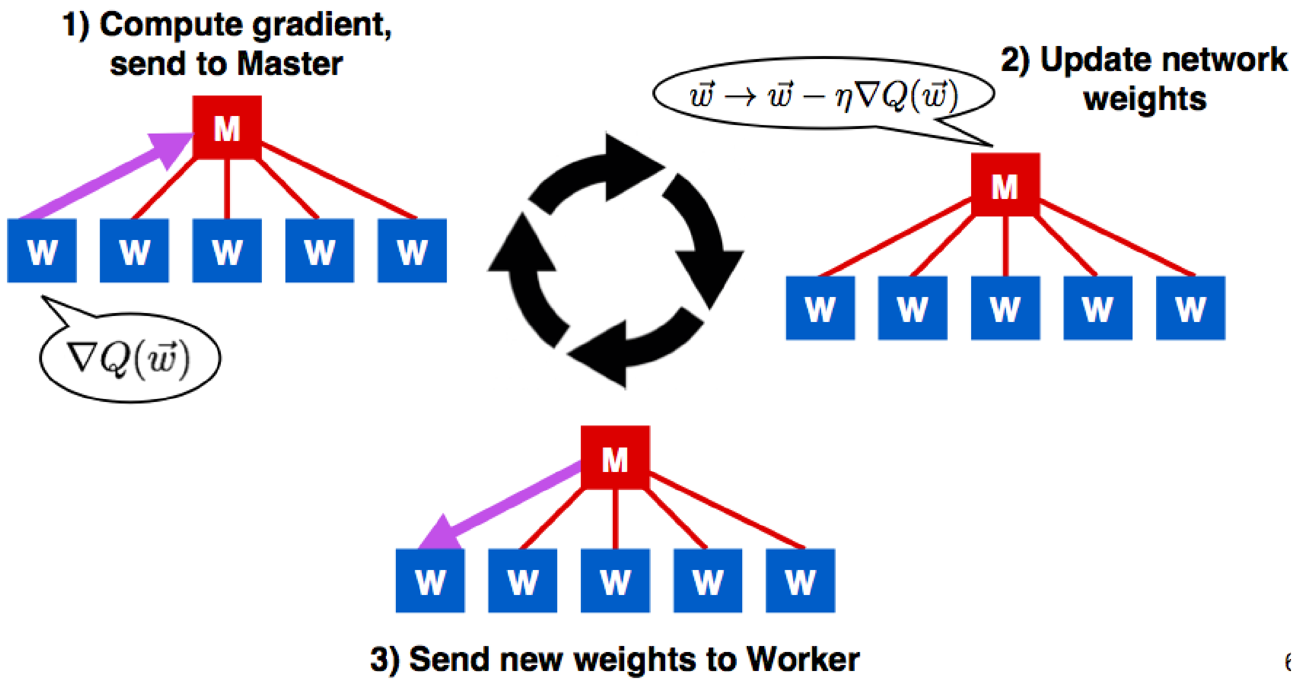
  Compute the gradient of one batch in parallel and update the model with the aggregated gradient. **Applicable to large sample ≡ large event**

- ➔ **Model distribution**

  Compute the gradient and updates of part of the model separately in chain. **Applicable to large model**

Data Distribution

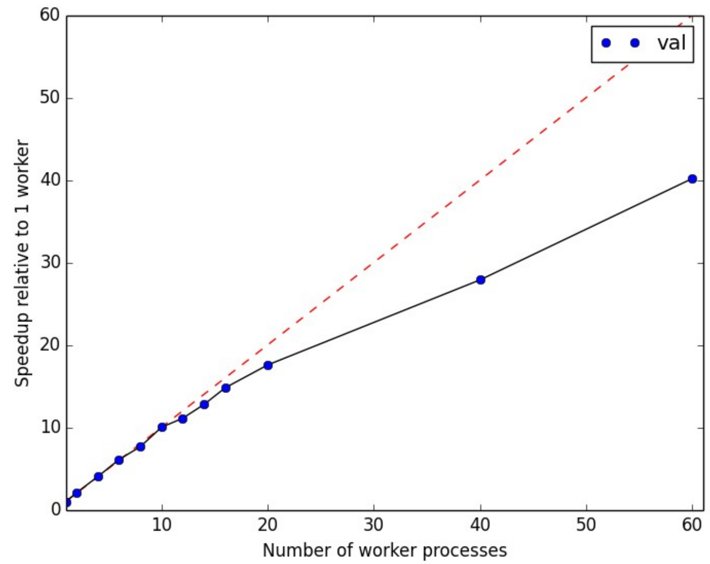
Data Distribution

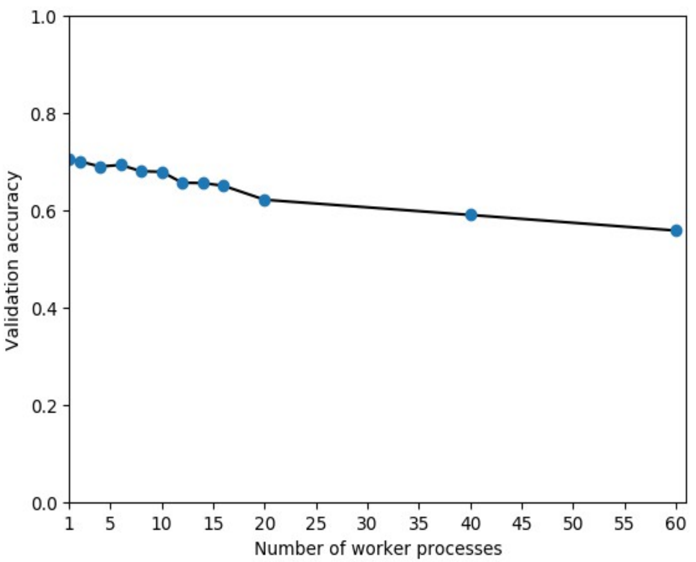


https://arxiv.org/abs/1712.05878

- Master node operates as parameter server
- Work nodes compute gradients
- Master handles gradients to update the central model
  - → downpour sgd https://tinyurl.com/ycfpwec5
  - → Elastic averaging sgd https://arxiv.org/abs/1412.6651

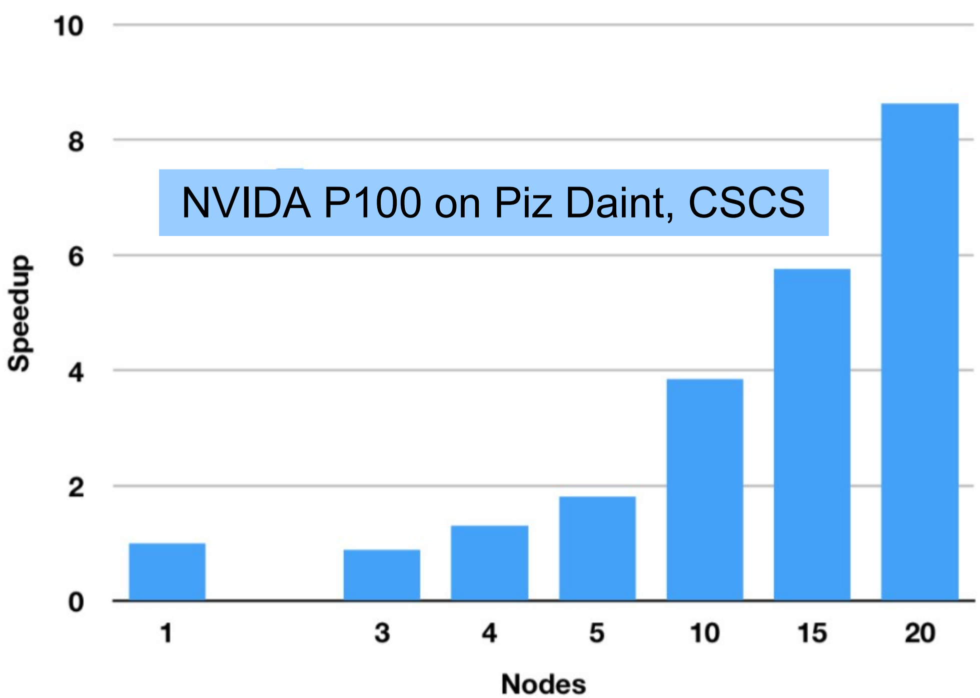
Performance on ANN

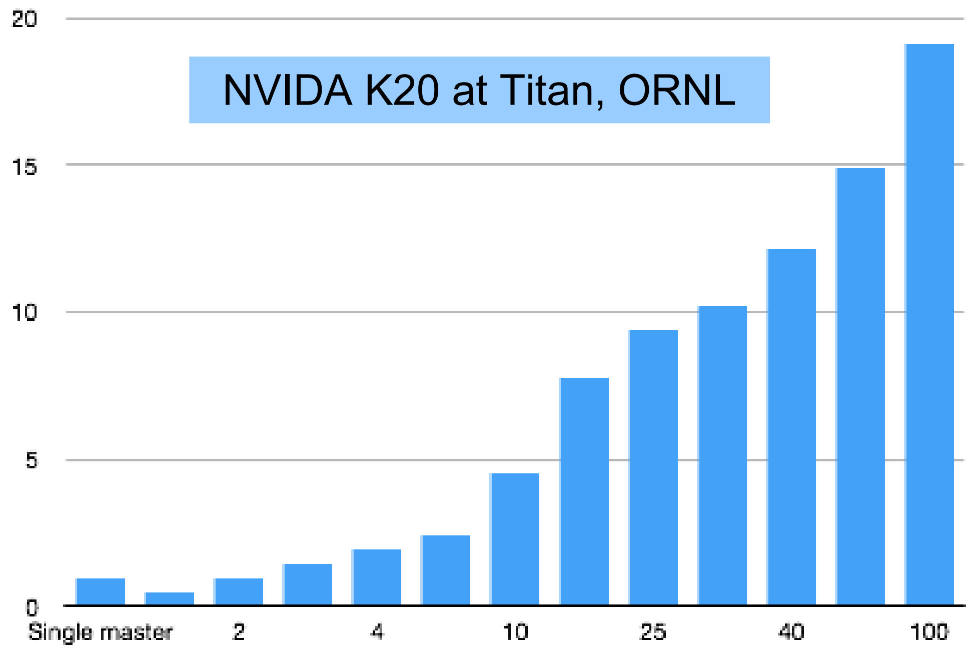




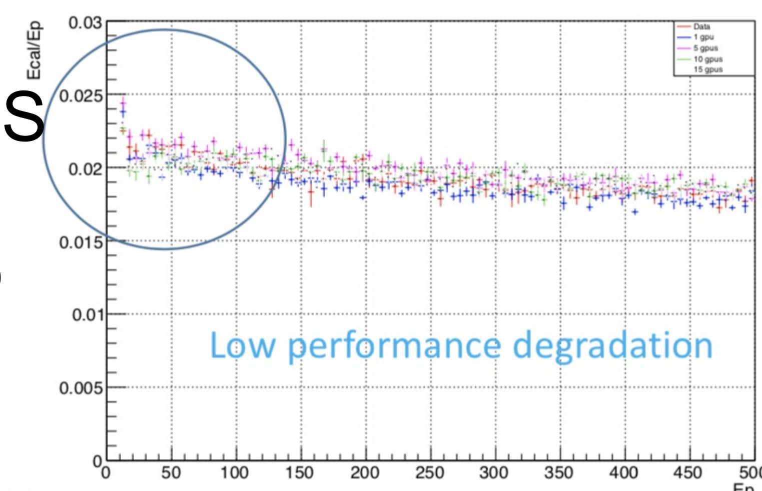
- Speed up in training recurrent neural networks on Piz Daint CSCS supercomputer
  - Linear speed up with up to ~20 nodes. Bottlenecks to be identified
  - Needs to compensate for staleness of gradients
- Similar scaling on servers with 8 GPUs
  - x7 speed up with students' work
- Gradient energy matching (https://arxiv.org/abs/1805.08469) implemented to mitigate staleness of gradients

Performance on GAN

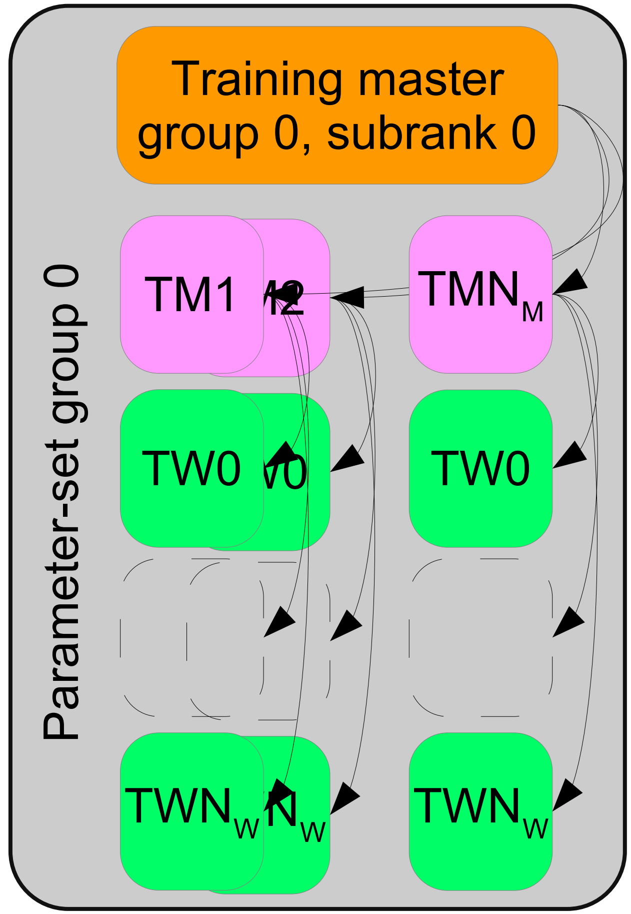




- Speed up in training generative adversarial networks on Piz Daint CSCS and Titan ORNL supercomputers
  - ➔ Using easgd algorithm with rmsprop
  - ➔ Speed up is not fully efficient. Bottlenecks to be identified



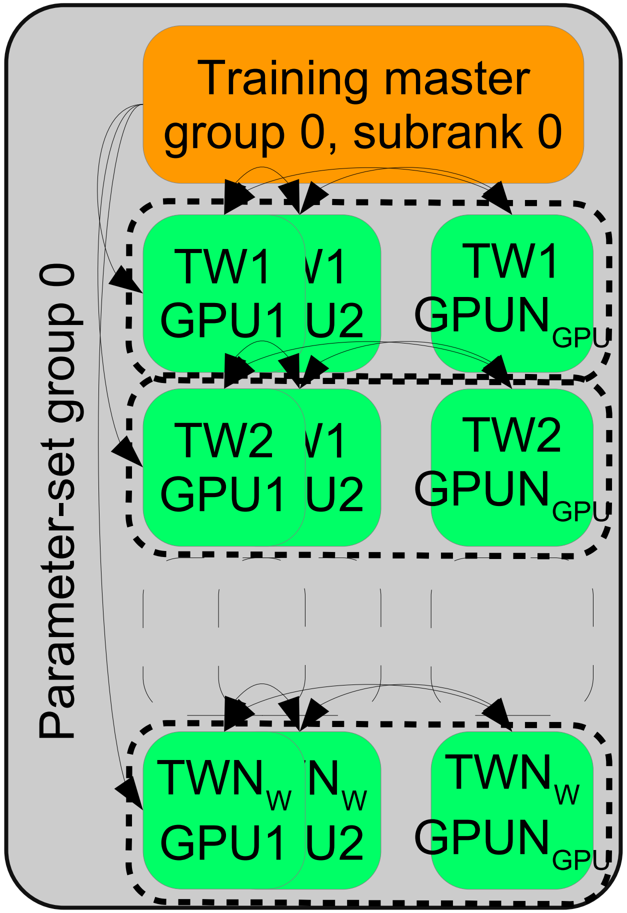
Sub-master Layout



- Putting workers in several groups
- Aim at spreading communication to the main master
- Need to strike a balance between staleness and update frequency

Gradient Distribution

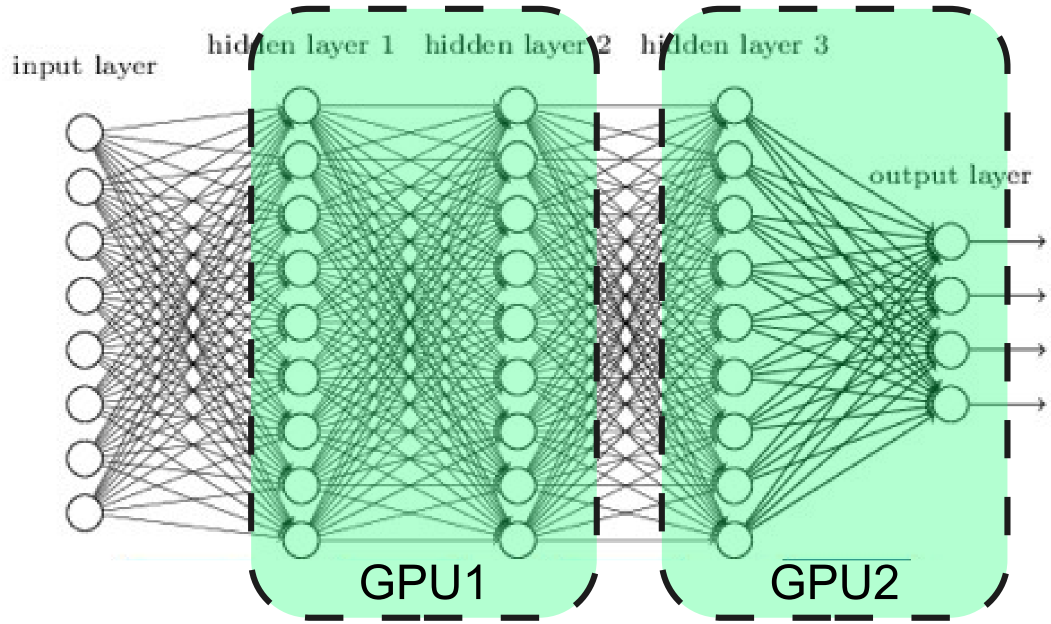
"all-reduce" Layout



- A logical worker is spawn over multiple mpi processes
- Communicator passed to **horovod** https://github.com/uber/horovod
- Private horovod branch to allow for group initialization/reset
- Nvidia NCCL enabled for fast GPU-GPU communication

Model Distribution

Intra-Node Model Parallelism



- Perform the forward and backward pass of sets of layers on different devices
- Require good device to device communication
- Utilize native tensorflow multi-device manager
- Aiming for machines with multi-gpu per node topology (summit)

Hyper-Parameters Optimization

Hyper-Parameters

- Various parameters of the model cannot be learned by gradient descent
  - ➢ Learning rate, batch size, number of layers, size of kernels, …

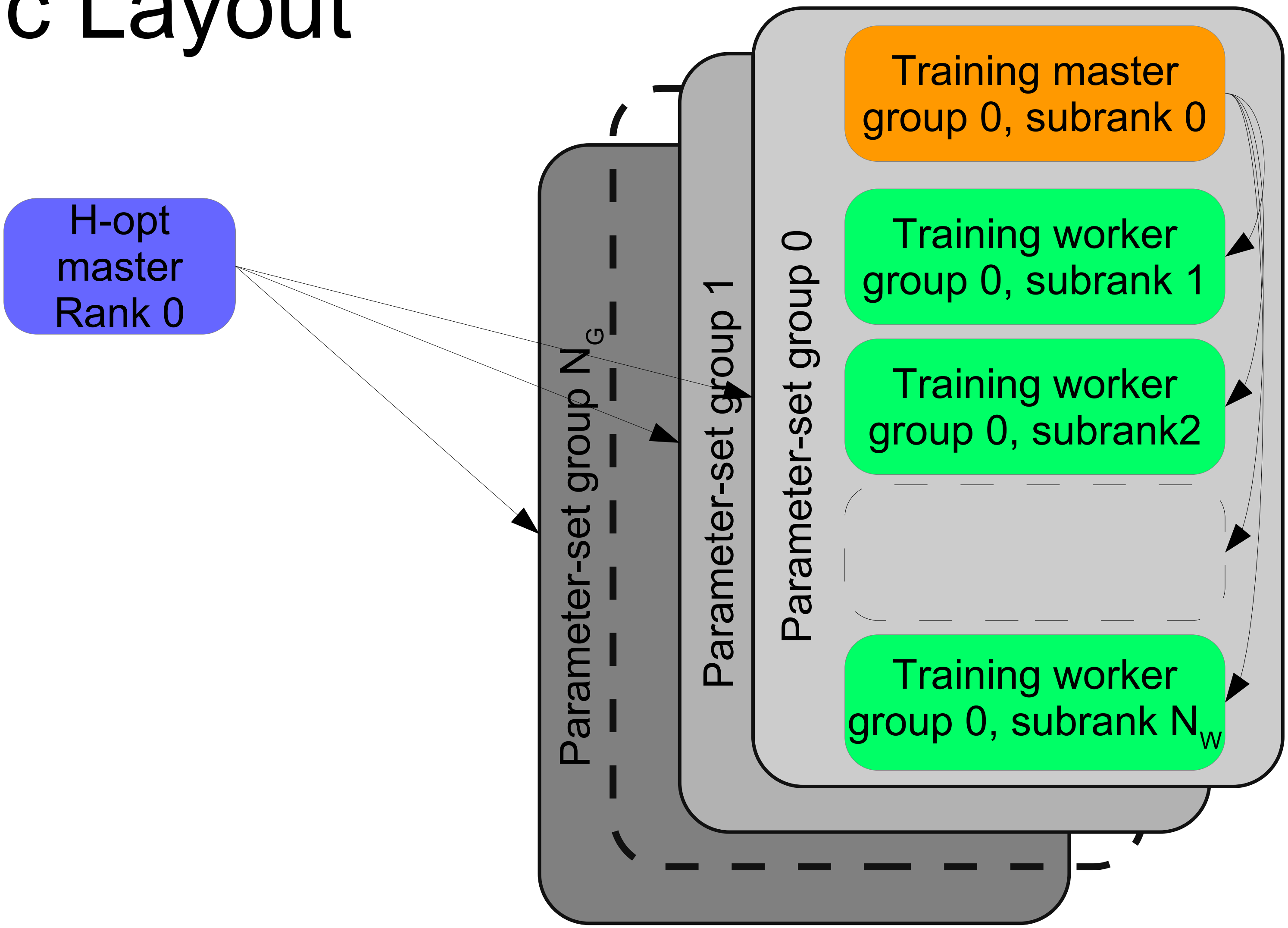
- Tuning to the right architecture is an “art”. Can easily spend a lot of time scanning many directions

- Full parameter scan is resource/time consuming.

➔ Hence looking for a way to reach the **optimum hyper-parameter** set for a provided **figure of merit** (the loss by default, but any other fom can work)

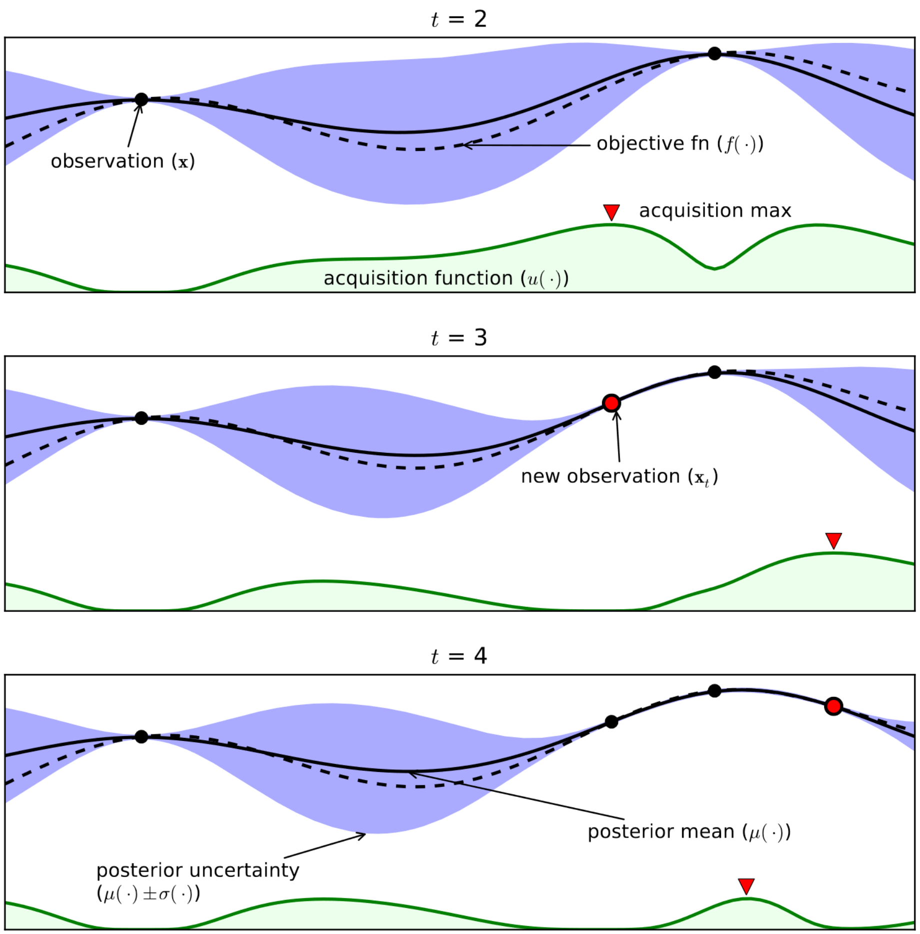
➔ Too optimization engine integrated
  - ➢ Bayesian optimization with gaussian processes prior
  - ➢ Evolutionary algorithm

Basic Layout



- One master process drives the hyper-parameter optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
  - N_W training workers

Bayesian Optimization



https://tinyurl.com/yc2phuaj

- Objective function is approximated as a multivariate gaussian
- Measurements provided one by one to improve knowledge of the objective function
- Next best parameter to test is determined from the acquisition function

- Using the python implementation from https://scikit-optimize.github.io

Evolutionary Algorithm

- Chromosomes are represented by the hyper-parameters
- Initial population taken at random in the parameter space
- Population is stepped through generations
  - Select the 20% fittest solutions
  - Parents of offspring selected by binary tournament based on fitness function
  - Crossover and mutate to breed offspring
- Alternative to bayesian opt. Indications that it works better for large number of parameters and non-smooth objective function

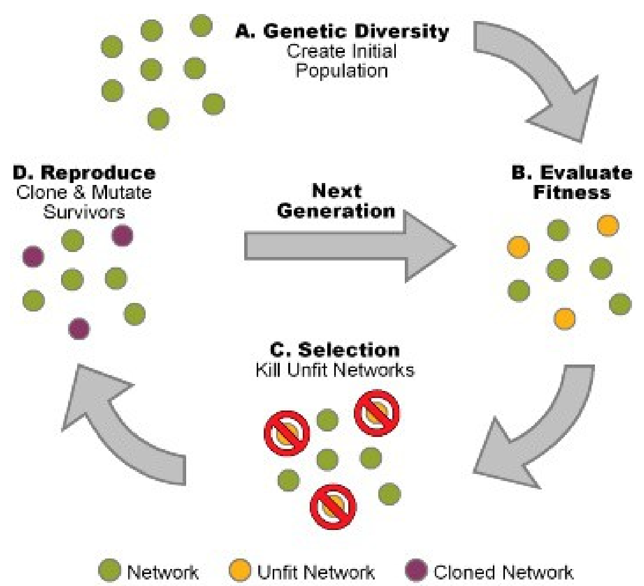
- Chromosome crossover:
  - Let Parent A be more fit than Parent B
  - For each parameter p, generate a random number r in (0, 1) to find p_{child}

$$p_{child} = (r)(p_{Parent\ A} - p_{Parent\ B}) + p_{Parent\ A}$$

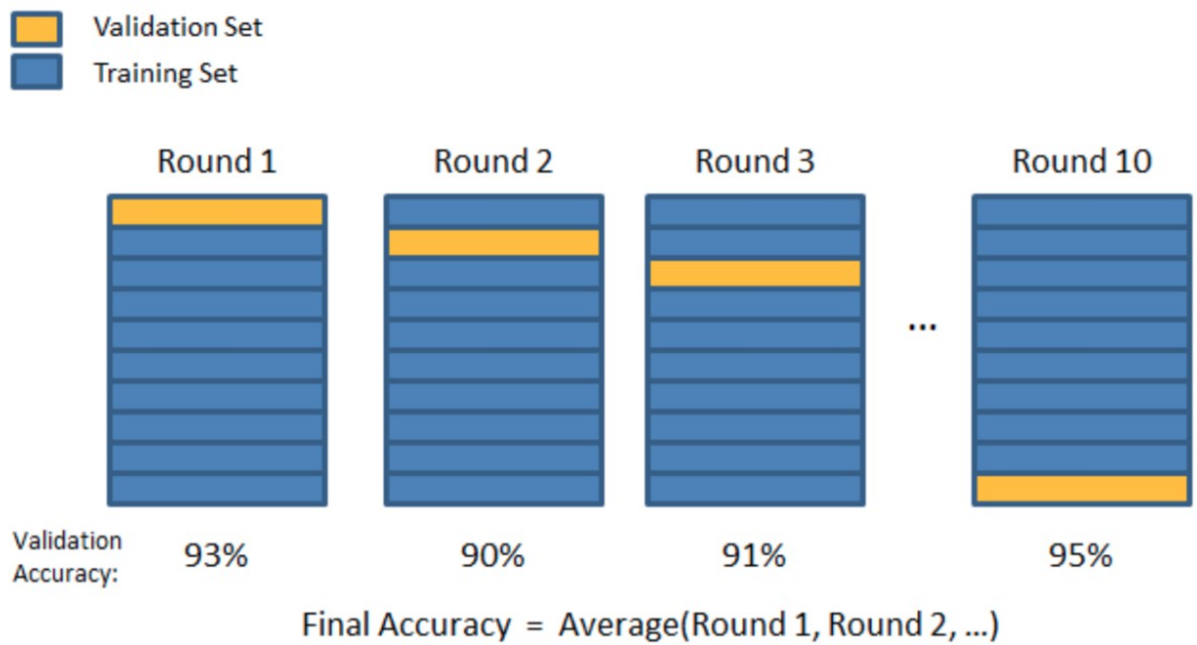
- Non-uniform mutation (Michalewicz):
  - In generation g out of a total G generations, for each parameter p in a child, generate random numbers $r_1, r_2 \in (0,1)$ to define a mutation m:

$$m = \left(1 - r_1^{\left(1-\frac{g}{G}\right)^3}\right) * \begin{cases}(p_{MAX} - p_{child}) & IF \quad r_2 > 0.5 \\ (p_{LOW} - p_{child}) & IF \quad r_2 \leq 0.5\end{cases}$$

$$p_{child} = p_{child} + m$$

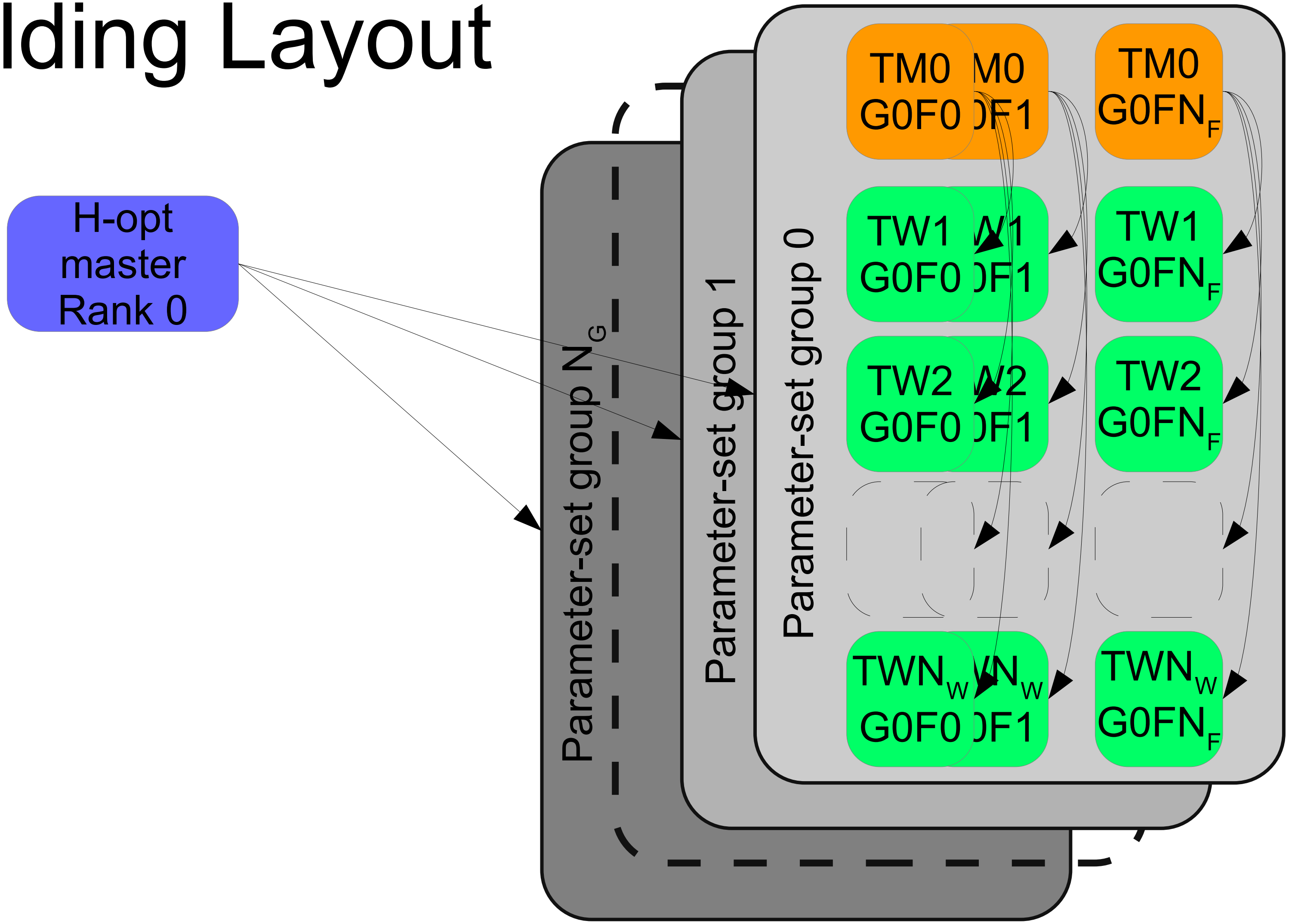


K-Folding Cross Validation



- Estimate the performance of multiple model training over different validation part of the training dataset
- Allows to take into account variance from multiple source (choice of validation set, choice of random initialization, ...)
- Crucial when comparing models performance
- Training on folds can proceed in parallel

K-folding Layout



- One master running the optimization. Receiving the average figure of merit over N_F folds of the data
  - N_G groups of nodes training on a parameter-set on simultaneously
    - N_F groups of nodes running one fold each

Summary and Outlook

Putting all Features Together

$N_{nodes} = 1 + N_G \times N_F \times (N_M \times N_W \times N_{GPU})$

N_G : # of concurrent hyper-parameter set tested
N_F : # of folds
N_M : # of masters
N_W : # of workers per master
N_{GPU} : # of nodes per worker (1node=1gpu)

Speed up and optimize models using thousand(s) of GPUs

Past, Present & Future

- Existing:
  - Support keras+tf and pytorch
  - GAN example
  - Data, gradient, model distribution
  - K-folding
  - Hyper-opt with BO, and EA

- Recently done:
  - ✓ Speed up performance on the master
  - ✓ Refactor code to one package https://github.com/vlimant/NNLO
  - ✓ Better interface for model, data, hyper-parameters
  - ✓ Proper logging
  - ✓ Checkpointing for training and optimization

- Still to be done:
  - ✗ Issue with distributed BatchNorm
  - ✗ Seamless support of GAN models (still a little adhoc)
  - ✗ Characterize speed up with updated master code
  - ✗ Characterize optimization speed-up / advantage
  - ✗ More documentation

Acknowledgements

- Part of this work was conducted on TACC under an allocation thanks to the Intel IPCC program.
- Part of this work was conducted on Titan at OLCF under the allocation csc291 (2018).
- Part of this work was conducted on Piz Daint at CSCS under the allocations d59 (2016) and cn01 (2018).
- Part of this work was conducted at "iBanks", the AI GPU cluster at Caltech. We acknowledge NVIDIA, SuperMicro and the Kavli Foundation for their support of "iBanks".
- Part of the team is funded by ERC H2020 grant number 772369

Extra Slides

Not mpi-opt

Cray ML plugin

MPI based. Synchronous SGD. TF1.4

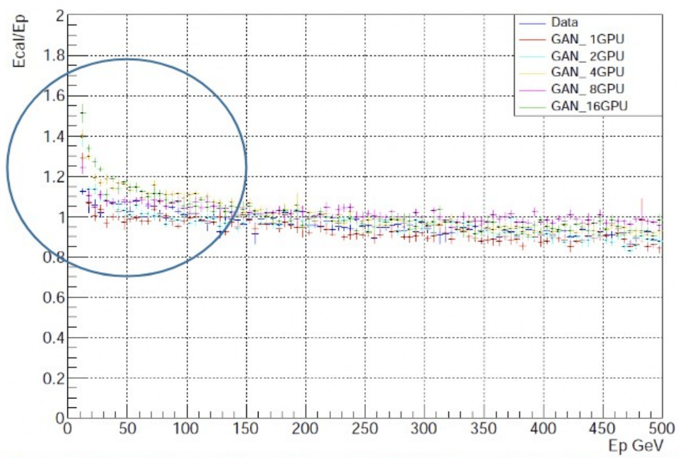
Optimal scaling through a large number of nodes

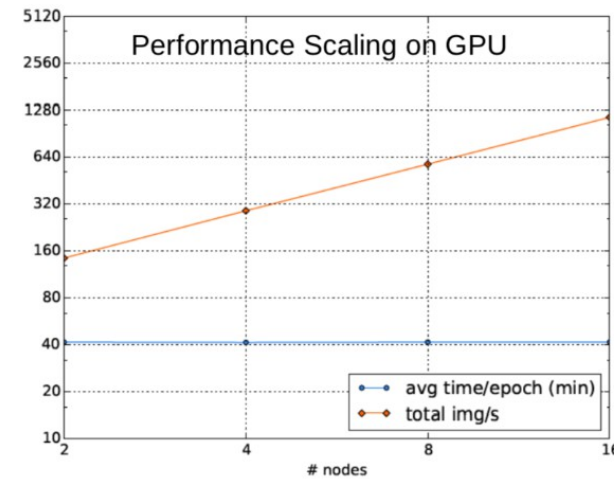
Observed performance degradation at low energy

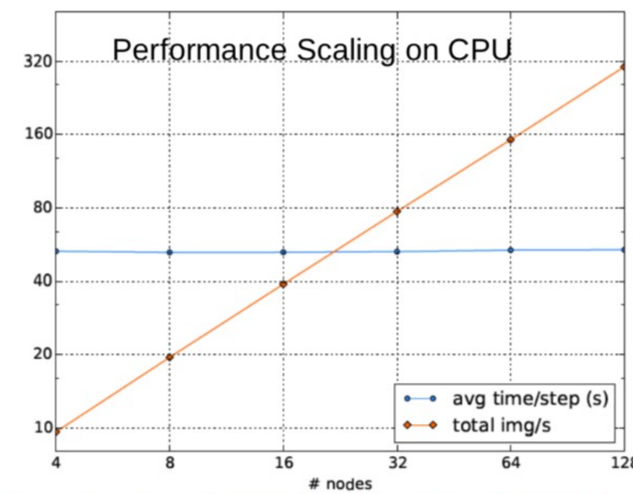
Possibly compensate by increasing learning rate

Work in progress

	GPU System	CPU System
Model	XC40/XC50	XC50
Computer nodes	Intel Xeon E5-2697 v4 @ 2.3GHz (18 cores, 64GB RAM) and NVIDIA Tesla P100 16GB	Two Intel Xeon Platinum 8160 @ 2.1GHz (2 x 24 cores, 192GB RAM)
Interconnect	Aries, Dragonfly network topology	Aries, Dragonfly network topology
Step	Epoch	Batch







16

Sofia V. @ https://sites.google.com/nvidia.com/ai-hpc

Not mpi-opt

Distributed training

Use keras 2.13 /Tensorflow 1.9 (Intel optimised)

- AVX512 –FMA-XLA support
- Intel® MKL-DNN (with 3D convolution support)

Optimised multicore utilisation

- inter_op_paralellism_threads/intra_op_paralellism threads

Horovod 0.13.4

- Synchronous SGD approach
- MPI_AllReduce

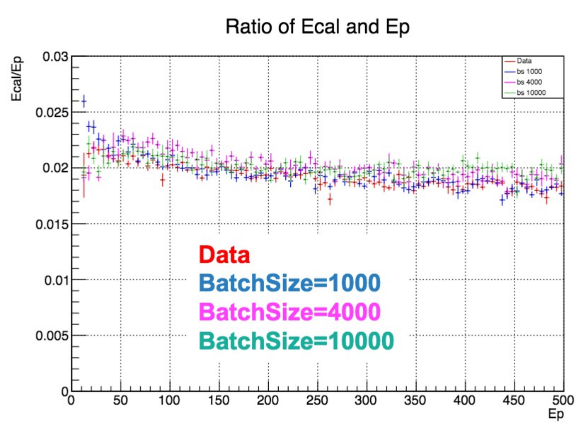
Run on TACC Stampede2 cluster:

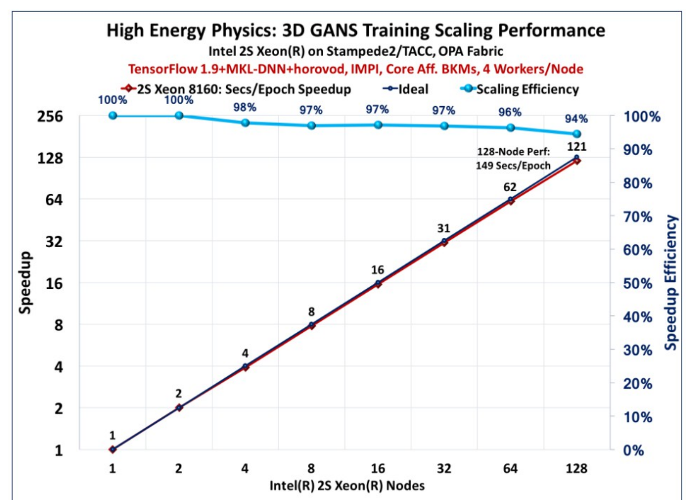
- Dual socket Intel Xeon 8160
- 2x 24 cores per node, 192 GB RAM
- Intel® Omni-Path Architecture

Test several MPI scheduling configurations

- 2,4, 8 processes per nodes.
- Best machine efficiency with 4 processes/node

Some performance degradation
Mostly in the low energy regions for large batchsize

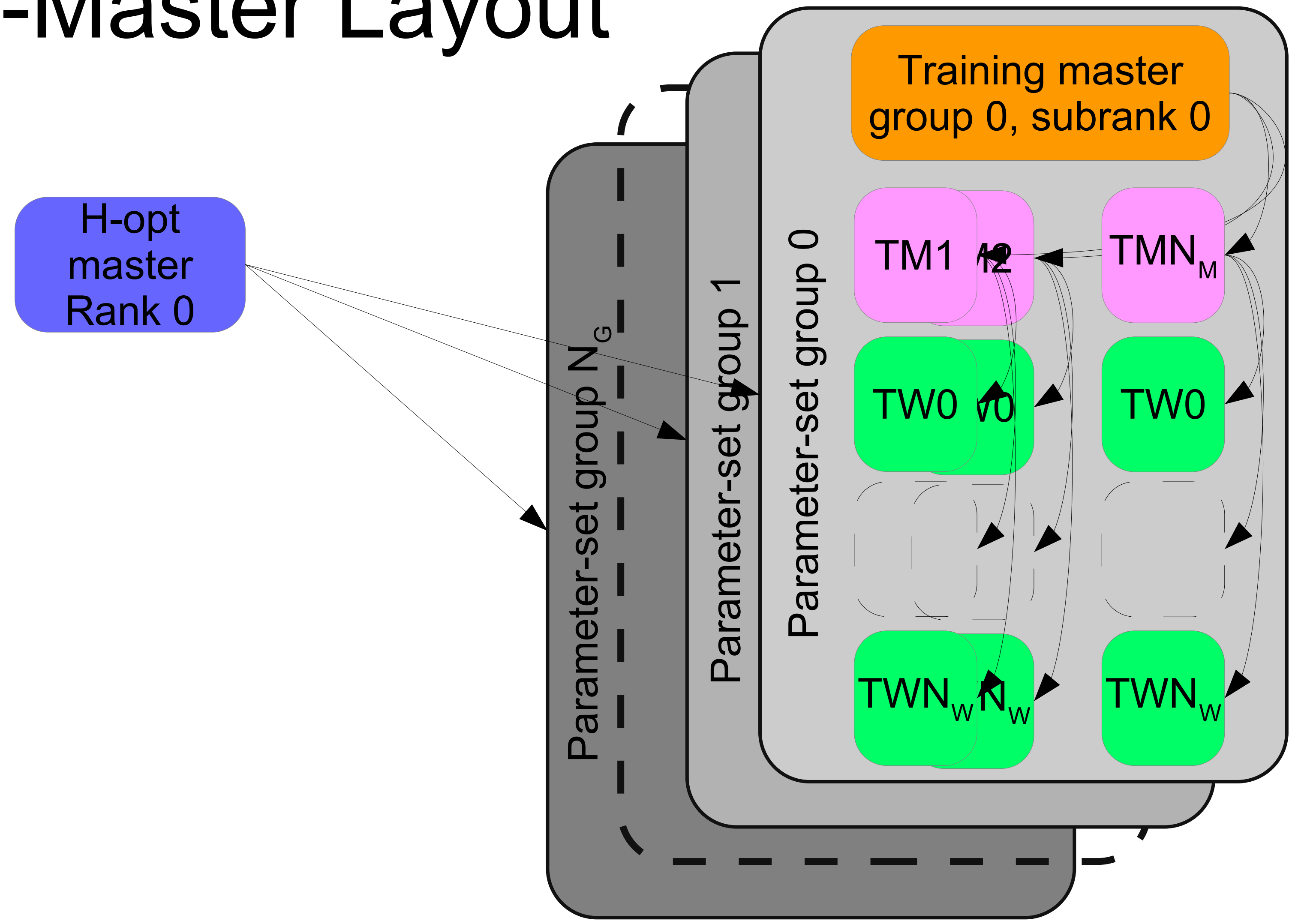




14

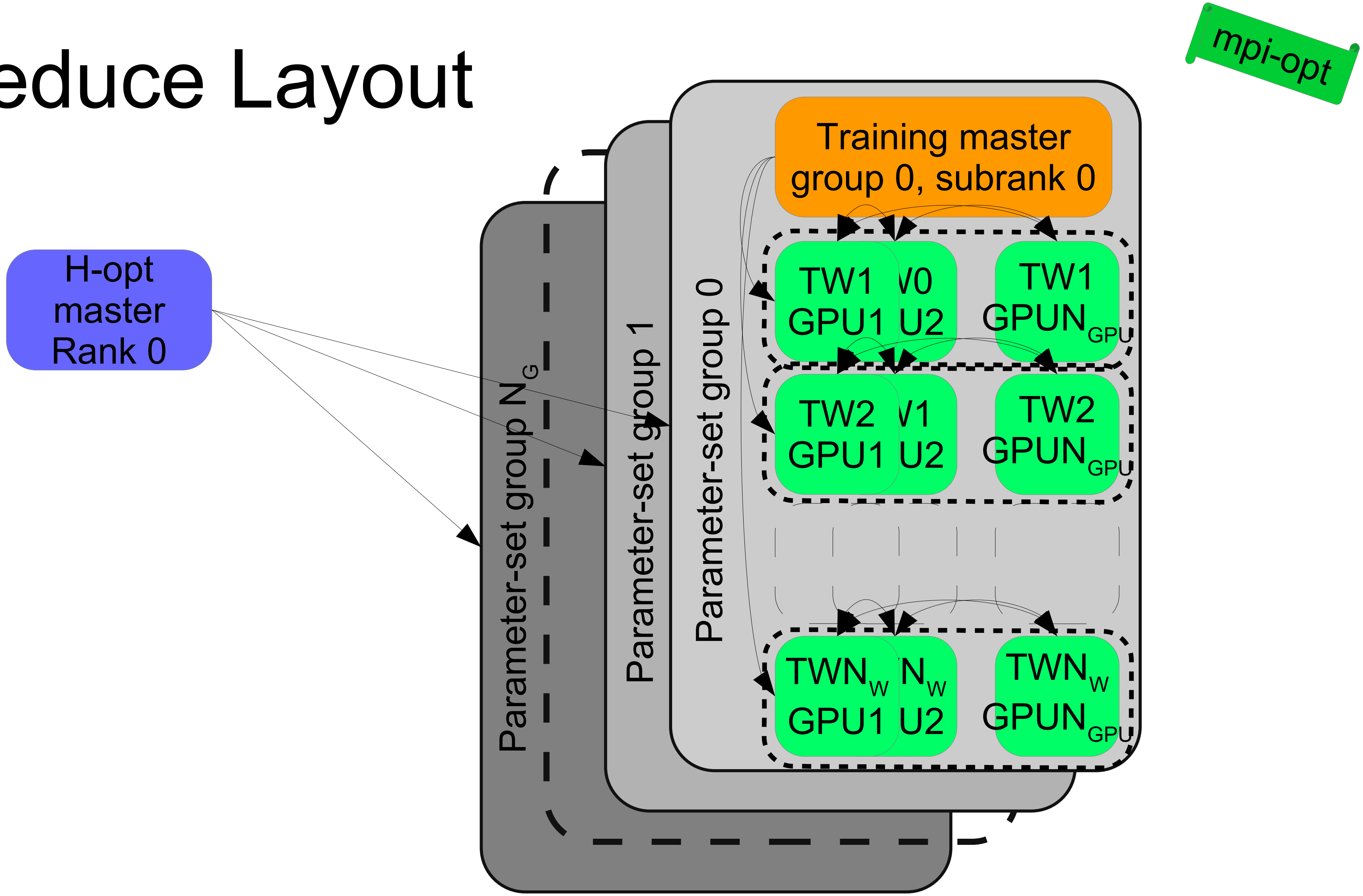
Sofia V. @ https://sites.google.com/nvidia.com/ai-hpc

Sub-Master Layout



- One master running the bayesian optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
    - N_M training sub-masters
      - N_W training workers

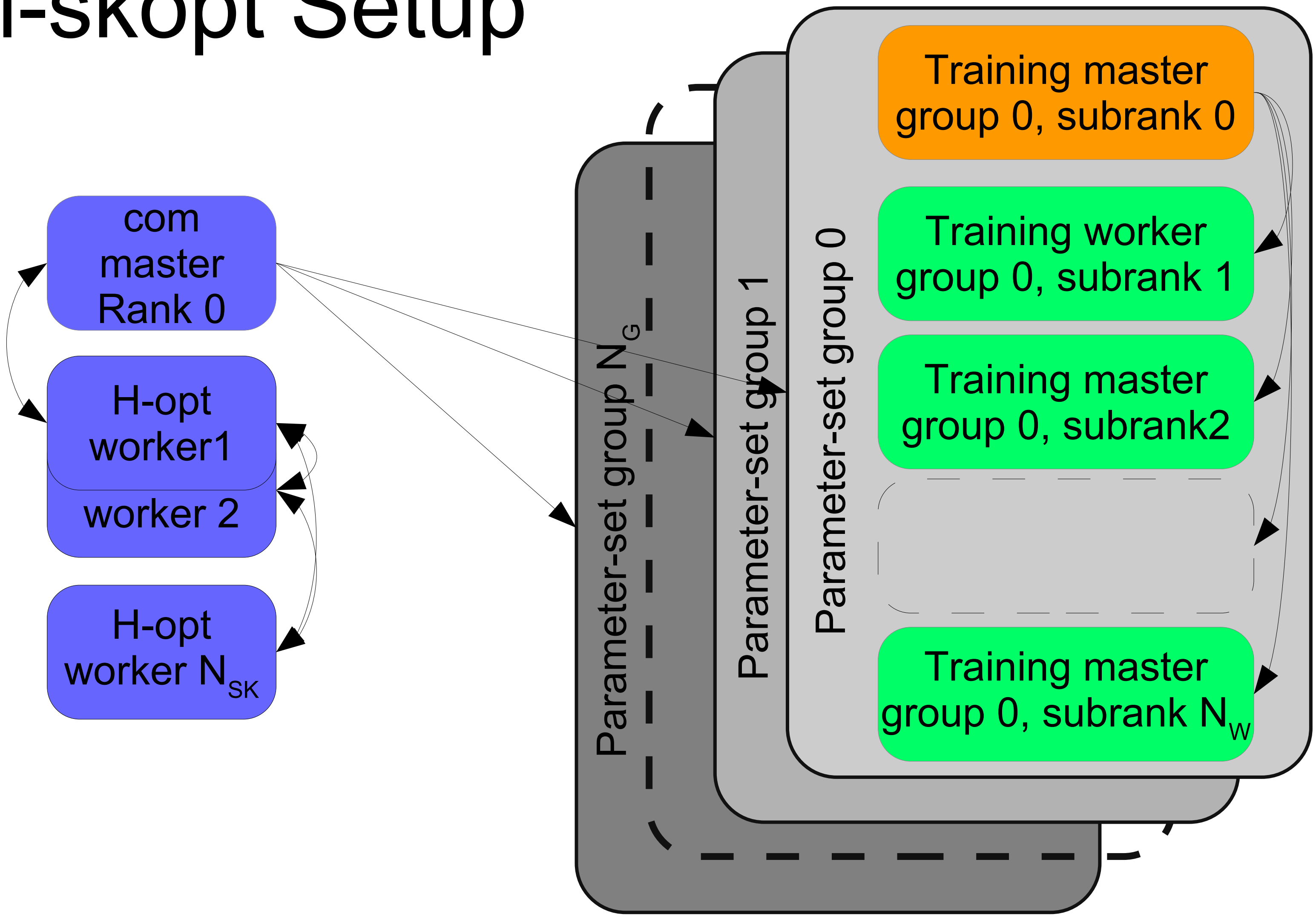
all-reduce Layout



- One master running the bayesian optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
  - N_W training worker groups
    - N_{GPU} used for each worker group (either nodes or gpu)

mpi-skopt Setup

mpi-opt



- One master running communication of parameter set
- N_{SK} workers running the bayesian optimization
- N_G groups of nodes training on a parameter-set on simultaneously
  - One training master
  - N_W training workers